Self-Stabilizing *(f,g)*-Alliances with Safe Convergence

Fabienne Carrier, Ajoy K. Datta, Stéphane Devismes, Lawrence L. Larmore, and Yvan Rivierre

Verimag Research Report n^o TR-2012-19

July 15, 2013

Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Self-Stabilizing *(f,g)*-Alliances with Safe Convergence

Fabienne Carrier, Ajoy K. Datta, Stéphane Devismes,
Lawrence L. Larmore, and Yvan Rivierre

July 15, 2013

Abstract

Given two functions f and g mapping nodes to non-negative integers, we give a silent self-stabilizing algorithm that computes a minimal (f, g)-alliance in an asynchronous network with unique node IDs, assuming that every node p has a degree at least $g(p)$ and satisfies $f(p) \geq g(p)$. Our algorithm is *safely converging* in the sense that starting from any configuration, it first converges to a (not necessarily minimal) (f, g)-alliance in at most four rounds, and then continues to converge to a minimal one in at most $5n + 4$ additional rounds, where n is the size of the network. Our algorithm is written in the shared memory model. It is proven assuming an unfair (distributed) daemon. Its memory requirement is $O(\log n)$ bits per process, and it takes $O(\Delta^3 n)$ steps to stabilize, where Δ is the degree of the network.

Keywords: Distributed Systems, Self-Stabilization, Safe Convergence, (f, g)-Alliance, Unfair Daemon

How to cite this report:

@techreport {TR-2012-19,
title = {Self-Stabilizing *(f,g)*-Alliances with Safe Convergence},
author = {Fabienne Carrier, Ajoy K. Datta, Stéphane Devismes,
Lawrence L. Larmore, and Yvan Rivierre},
institution = {{Verimag} Research Report},
number = {TR-2012-19},
year = {2012}
}

1 Introduction

Self-stabilization [1] is a versatile technique to withstand *any* transient fault in a distributed system. Informally, a distributed algorithm is self-stabilizing if, after transient faults hit the system and place it in some arbitrary configuration, the system recovers without external (*e.g.*, human) intervention in finite time. Thus, self-stabilization makes no hypothesis on the nature or extent of transient faults that could hit the system, and recovers from the effects of those faults in a unified manner. However, self-stabilization has some drawbacks; perhaps the main one is *temporary loss of safety*, *i.e.*, after the occurrence of transient faults, there is a finite period of time — called the *stabilization phase* — before the system returns to a legitimate configuration. During this phase, there is no guarantee of safety. Several approaches have been introduced to offer more stringent guarantees during the stabilization phase, *e.g.*, *fault-containment* [2], *superstabilization* [3], *time-adaptivity* [4], and *safe convergence* [5].

We consider here the notion of *safe convergence*. The main idea behind this concept is the following: For a large class of problems, it is often hard to design self-stabilizing algorithms that guarantee small stabilization time, even after few transient faults [6]. Large stabilization time is usually due to strong specifications that a legitimate configuration must satisfy. The goal of a *safely converging self-stabilizing algorithm* is to first quickly converge ($O(1)$ rounds is usually expected) to a *feasible* legitimate configuration, where a minimum quality of service is guaranteed. Once such a feasible legitimate configuration is reached, the system continues to converge to an *optimal* legitimate configuration, where more stringent conditions are required. Safe convergence is especially interesting for self-stabilizing algorithms that compute optimized data structures, *e.g.*, minimal dominating sets [5], approximation of the minimum weakly connected dominating set [7], and approximately minimum connected dominating set [8].

We consider the (f,g)-*alliance problem*. Let $G=(V,E)$ be an undirected graph and f, g two functions mapping nodes to non-negative integers. For every node $p \in V$, $\mathcal{N}_p$ (resp. δ_p) denotes the set of neighbors (resp. the degree) of p in G. A subset of nodes $A \subseteq V$ is an (f,g)-*alliance* of G if and only if

$$(\forall p \in V \setminus A, |\mathcal{N}_p \cap A| \geq f(p)) \wedge (\forall p \in A, |\mathcal{N}_p \cap A| \geq g(p))$$

Moreover, A is *minimal* if and only if no proper subset of A is an (f,g)-alliance of G. The (f,g)-alliance problem is a generalization of several problems that are of interest in distributed computing. Consider any subset S of nodes:

1. S is a (minimal) dominating set if and only if S is a (minimal) $(1,0)$-alliance;
2. more generally, S is a (minimal) k-dominating set[1] if and only if S is a (minimal) $(k,0)$-alliance;
3. S is a (minimal) k-tuple dominating set if and only if S is a (minimal) $(k,k-1)$-alliance;
4. S is a (minimal) global defensive alliance if and only if S is a (minimal) $(f,0)$-alliance, such that $\forall p \in V$, $f(p) = \lceil \delta_p/2 \rceil$;
5. S is a (minimal) global offensive alliance if and only if S is a (minimal) $(1,g)$-alliance, such that $\forall p \in V, g(p) = \lceil \delta_p/2 \rceil$.

Note that (f,g)-alliances also have applications in the field of population protocols [9], or server allocation in computer networks [10].

1.1 Our Contribution

We give a silent self-stabilizing algorithm, $\mathcal{MA}(f,g)$, that computes a minimal (f,g)-alliance in an asynchronous network with unique node IDs, where f and g are integer-valued functions on nodes, such that $f(p) \geq g(p)$ and $\delta_p \geq g(p)$ for all p.[2]

Given two functions f,g mapping nodes to non-negative integers, we say $f \geq g$ if and only if $\forall p \in V, f(p) \geq g(p)$. We remark that the class of minimal (f,g)-alliances with $f \geq g$ generalizes the classes

[1] In the literature, *k-dominating set* had multiple definitions. Here, we consider the definition that S is a k-dominating set if and only if every node that is not in S has at least k neighbors in S.

[2] We assume that $\delta_p \geq g(p)$ to ensure that an (f,g)-alliance always exists.

of minimal dominating sets, k-dominating sets, k-tuple dominating sets, and global defensive alliance problems. However, minimal global offensive alliances do not belong to this class.

Our algorithm $\mathcal{MA}(f,g)$ is *safely converging* in the sense that starting from any configuration, it first converges to a (not necessarily minimal) (f,g)-alliance in at most four rounds, and then continues to converge to a minimal one in at most $5n+4$ additional rounds, where n is the size of the network. Our algorithm is written in the shared memory model, and is proven assuming an unfair (distributed) daemon, the weakest daemon of this model. $\mathcal{MA}(f,g)$ uses $O(\log n)$ bits per process, and stabilizes to a terminal (legitimate) configuration in $O(\Delta^3 n)$ steps, where Δ is the degree of the network. Finally, $\mathcal{MA}(f,g)$ does not need any knowledge of any bound on global parameters of the network (such as its size or its diameter).

1.2 Related Work

The (f,g)-alliance problem is introduced in [11]. In the same paper, the authors give several distributed algorithms for that problem and its variants, but none of them is self-stabilizing. To the best of our knowledge, this has been the only publication on (f,g)-alliances up to now. However, there have been results on particular instances of (minimal) (f,g)-alliances, *e.g.*, [5, 12, 13, 14]. All of these consider arbitrary identified networks; however a safely converging solution is given only in [5]. Srimani and Xu [12] give a self-stabilizing algorithm to compute a minimal global defensive alliance in $O(n^3)$ steps; however, they assume a central daemon. Turau [13] gives a self-stabilizing algorithm to compute a minimal dominating set in $9n$ steps, assuming an unfair (distributed) daemon. Wang *et al* [14] give a self-stabilizing algorithm to compute a minimal k-dominating set in $O(n^2)$ steps, assuming a central daemon. A safely converging self-stabilizing algorithm is given in [5] for computing a minimal dominating set. The algorithm first computes a (not necessarily minimal) dominating set in $O(1)$ rounds and then safely stabilizes to a *minimal* dominating set in $O(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network. However, they assume a synchronous daemon.

1.3 Roadmap

In the next section we describe our model of computation and give some basic definitions. We define our algorithm $\mathcal{MA}(f,g)$ in Section 3. In Section 4, we show the correctness of $\mathcal{MA}(f,g)$ and analyze its complexity. We write concluding remarks and perspectives in Section 5.

2 Preliminaries

2.1 Distributed Systems

We consider distributed systems of n processes with *unique* IDs. By an abuse of notation, we identify a process with its ID whenever convenient. Each process p can directly communicate with a subset $\mathcal{N}_p$ of other processes, called its *neighbors*. We assume that if $q \in \mathcal{N}_p$, then $p \in \mathcal{N}_q$. For every process p, $\delta_p = |\mathcal{N}_p|$ is the *degree of* p. We assume that $\delta_p \geq g(p)$ for every process p. Let $\Delta = \max_{p \in V} \delta_p$ be the degree of the network. The topology of the system is a simple undirected graph $G = (V,E)$, where V is the set of processes and E is a set of edges representing (direct) communication relations.

2.2 Computational Model

We assume the *shared memory model* of computation introduced by Dijkstra [1], where each process communicates with its neighbors using a finite set of *locally shared variables*, henceforth called simply *variables*. Each process can read its own variables and those of its neighbors, but can write only to its own variables. Each process operates according to its (local) *program*. We define a *(distributed) algorithm* to be a collection of n *programs*, each operating on a single process. The program of each process is a finite ordered set of actions, where the ordering defines *priority*. This priority is the order of appearance of actions in the text of the program. A process p is not enabled to execute any action if it is enabled to execute an action of higher priority. Let $\mathcal{A}$ be a distributed algorithm, consisting of a local program $\mathcal{A}(p)$ for each process p. Each action in $\mathcal{A}(p)$ is of the following form:

$$\langle \textit{label} \rangle \ :: \ \langle guard \rangle \ \rightarrow \ \langle \textit{statement} \rangle$$

Labels are only used to identify actions. The *guard* of an action in $\mathcal{A}(p)$ is a Boolean expression involving the variables of p and its neighbors. The *statement* of an action in $\mathcal{A}(p)$ updates some variables of p. The *state* of a process in $\mathcal{A}$ is defined by the values of its variables in $\mathcal{A}$. A *configuration* of $\mathcal{A}$ is an instance of the states of processes in $\mathcal{A}$. $\mathcal{C}_\mathcal{A}$ is the set of all possible configurations of $\mathcal{A}$. (When there is no ambiguity, we omit the subscript $\mathcal{A}$.) An action can be executed only if its guard evaluates to *true*; in this case, the action is said to be *enabled*. A process is said to be enabled if at least one of its actions is enabled. We denote by *Enabled*(γ) the subset of processes that are enabled in configuration γ. When the configuration is γ and *Enabled*$(\gamma) \neq \emptyset$, a *daemon*[3] (scheduler) selects a non-empty set $\mathcal{X} \subseteq$ *Enabled*(γ); then every process of $\mathcal{X}$ *atomically* executes its highest priority enabled action, leading to a new configuration γ', and so on. The transition from γ to γ' is called a *step* (of $\mathcal{A}$). The possible steps induce a binary relation over configurations of $\mathcal{A}$, denoted by $\mapsto$. An *execution* of $\mathcal{A}$ is a maximal sequence of its configurations $e = \gamma_0\gamma_1 \dots \gamma_i \dots$ such that $\gamma_{i-1} \mapsto \gamma_i$ for all $i > 0$. The term "maximal" means that the execution is either infinite, or ends at a *terminal* configuration in which no action of $\mathcal{A}$ is enabled at any process. As we saw previously, each step from a configuration to another is driven by a daemon. In this paper we assume the daemon is *unfair*; *i.e.*, the daemon might never permit an enabled process to execute unless it is the only enabled process.

We say that a process p is *neutralized* in the step $\gamma_i \mapsto \gamma_{i+1}$ if p is enabled in γ_i and not enabled in γ_{i+1}, but does not execute any action between these two configurations. Neutralization of a process can be caused by the following situation: at least one neighbor of p changes its state between γ_i and γ_{i+1}, and this change makes the guards of all actions of p false.

To evaluate time complexity, we use the notion of *round*. The first round of an execution e, noted e', is the minimal prefix of e in which every process that is enabled in the initial configuration either executes an action or becomes neutralized. Let e'' be the suffix of e starting from the last configuration of e'. The second round of e is the first round of e'', and so forth.

2.3 Self-Stabilization, Silence, and Safe Convergence

Let $\mathcal{A}$ be a distributed algorithm. Let P be a predicate over $\mathcal{C}$. $\mathcal{A}$ is *self-stabilizing w.r.t.* P if and only if there exists a non-empty subset $\mathcal{S}_P$ of $\mathcal{C}$ such that:

1. $\forall \gamma \in \mathcal{S}_P$, $P(\gamma)$ (*Correction*);
2. for each possible step $\gamma \mapsto \gamma'$ of $\mathcal{A}$, $\gamma \in \mathcal{S}_P \Rightarrow \gamma' \in \mathcal{S}_P$ (*Closure*);
3. each execution of $\mathcal{A}$ (starting from an arbitrary configuration) contains a configuration of $\mathcal{S}_P$ (*Convergence*).

The configurations of $\mathcal{S}_P$ are said to be *legitimate*, and other configurations are called *illegitimate*.

$\mathcal{A}$ is *silent* if all its executions are finite [15]. To show that $\mathcal{A}$ is silent and self-stabilizing *w.r.t.* P, it is sufficient to show that

1. all executions of $\mathcal{A}$ are finite and
2. all terminal configurations of $\mathcal{A}$ satisfy P.

Let P_1 and P_2 be two predicates over $\mathcal{C}$ such that $\forall \gamma \in \mathcal{C}$, $P_2(\gamma) \Rightarrow P_1(\gamma)$. $\mathcal{A}$ is *safely converging self-stabilizing w.r.t.* (P_1, P_2) if and only if the following three properties hold:

1. $\mathcal{A}$ is *self-stabilizing w.r.t.* P_1;
2. $\mathcal{A}$ is *self-stabilizing w.r.t.* P_2; and

[3] The daemon realizes the asynchrony of the system.

3. every execution of $\mathcal{A}$ starting from a configuration of $\mathcal{S}_{P_1}$ eventually reaches a configuration of $\mathcal{S}_{P_2}$, where $\mathcal{S}_{P_1}$ and $\mathcal{S}_{P_2}$ are respectively the sets of legitimate configurations for P_1 and P_2 (*Safe Convergence*).

The configurations of $\mathcal{S}_{P_1}$ are said to be *feasible legitimate*. The configurations of $\mathcal{S}_{P_2}$ are said to be *optimal legitimate*.

Assume that $\mathcal{A}$ is *safely converging self-stabilizing w.r.t.* (P_1, P_2). The *first convergence time* is the maximum time to reach a feasible legitimate configuration, starting from any configuration. The *second convergence time* is the maximum time to reach an optimal legitimate configuration, starting from any feasible legitimate configuration. The *stabilization time* is the sum of the first and second convergence times.

2.4 Minimality and 1-Minimality of (f, g)-alliances

We recall that an (f, g)-alliance A of a graph G is *minimal* if and only if no proper subset of A is an (f, g)-alliance. Then, A is 1-*minimal* if and only if $\forall p \in A$, $A \setminus \{p\}$ is not an (f, g)-alliance. Surprisingly, a 1-*minimal* (f, g)-alliance is not necessarily a *minimal* (f, g)-alliance, [11]. However, we have the following property:

Property 1 [11] *Given two functions f and g mapping nodes to non-negative integers, we have:*

1. *Every minimal (f, g)-alliance is a 1-minimal (f, g)-alliance, and*
2. *if $f \geq g$, every 1-minimal (f, g)-alliance is a minimal (f, g)-alliance.*

3 The Algorithm

The formal code of $\mathcal{MA}(f, g)$ is given in Algorithm 1. Given the input functions f and g, $\mathcal{MA}(f, g)$ computes a single output for each process p: the Boolean $p.inA$. In any configuration γ, we define the set $A_\gamma = \{p \in V, p.inA\}$. (We omit the subscript γ when it is clear from the context.) And, if γ is terminal, then A_γ is a 1-*minimal* (f, g)-*alliance*, and consequently, if $f \geq g$, A_γ is a *minimal* (f, g)-*alliance*.

During an execution, a process may need to leave or join A. Then, the basic idea of safe convergence is that it should be more difficult for a process to leave A than to join it. Indeed, this permits quick recovery to a configuration in which A is an (f, g)-alliance, but not necessarily a minimal one.

3.1 Leaving A

Action `Leave` allows a process to leave A. To obtain 1-minimality, we allow a process p to leave A if

Requirement 1: p will have enough neighbors in A (*i.e.*, at least $f(p)$) once it has left, and

Requirement 2: each $q \in \mathcal{N}_p$ will still have enough neighbors in A (*i.e.*, at least $g(q)$ or $f(q)$, depending on whether q is in A) once p has been deleted from A.

Ensuring Requirement 1. To maintain Requirement 1, we implement our algorithm in such a way that deletion from A is *locally sequential*, *i.e.*, during a step, at most one process can leave A in the neighborhood of each process p (including p itself). Using this locally sequential mechanism, if a process p wants to leave A, it must first verify that $\texttt{NbA}(p) = |\{q \in \mathcal{N}_p, q.inA\}|$ is greater or equal to $f(p)$ before leaving A. Hence, if p actually leaves A, it is the only one in its neighborhood allowed to do that and, consequently, Requirement 1 still holds once p has left A.

The locally sequential mechanism is implemented using a neighbor pointer $p.choice$ at each process p, which takes value in $\mathcal{N}_p \cup \{\perp\}$: $p.choice = \perp$ means that p authorizes no neighbor to leave A; while $p.choice = q \in \mathcal{N}_p$ means that p authorizes its neighbor q to leave A. The value of $p.choice$ is maintained using Action `Vote`, which will be detailed later.

Algorithm 1 $\mathcal{MA}(f, g)$, code for each process p

Variables:

$p.inA$: Boolean
$p.busy$: Boolean
$p.choice \in \mathcal{N}_p \cup \{\bot\}$
$p.nbA \in [0..\delta_p]$

Macros:

$\texttt{NbA}(p) = |\{q \in \mathcal{N}_p, q.inA\}|$
$\texttt{Cand}(p) = \{q \in \mathcal{N}_p, q.inA \wedge \neg q.busy\}$
$\texttt{MinCand}(p) = \min(\texttt{Cand}(p) \cup \{\infty\})$
$\texttt{ChosenCand}(p) =$ **if** $\texttt{Cand}(p) \neq \emptyset \wedge \texttt{HasExtra}(p) \wedge (\texttt{IamCand}(p) \Rightarrow \texttt{MinCand}(p) < p)$
  then $\texttt{MinCand}(p)$
  else $\bot$
$\texttt{Choice}(p) =$ **if** $p.choice = \bot$
  then $\texttt{ChosenCand}(p)$
  else $\bot$

Predicates:

$\texttt{IsMissing}(p) \equiv \exists q \in \mathcal{N}_p, (\neg q.inA \wedge q.nbA < f(q)) \vee (q.inA \wedge q.nbA < g(q))$
$\texttt{IsExtra}(p) \equiv \forall q \in \mathcal{N}_p, (\neg q.inA \Rightarrow q.nbA > f(q)) \wedge (q.inA \Rightarrow q.nbA > g(q))$
$\texttt{HasExtra}(p) \equiv (\neg p.inA \Rightarrow \texttt{NbA}(p) > f(p)) \wedge (p.inA \Rightarrow \texttt{NbA}(p) > g(p))$
$\texttt{IsBusy}(p) \equiv \texttt{NbA}(p) < f(p) \vee \neg\texttt{IsExtra}(p)$
$\texttt{IamCand}(p) \equiv p.inA \wedge \neg\texttt{IsBusy}(p)$
$\texttt{MustJoin}(p) \equiv \neg p.inA \wedge (\texttt{NbA}(p) < f(p) \vee \texttt{IsMissing}(p)) \wedge (\forall q \in \mathcal{N}_p, q.choice \neq p)$
$\texttt{CanLeave}(p) \equiv p.inA \wedge \texttt{NbA}(p) \geq f(p) \wedge (\forall q \in \mathcal{N}_p, q.choice = p) \wedge p.choice = \bot$

Actions:

`Join`	::	$\texttt{MustJoin}(p)$	$\to$	$p.inA \leftarrow true$ $p.choice \leftarrow \bot$ $p.nbA \leftarrow \texttt{NbA}(p)$
`Vote`	::	$p.choice \neq \texttt{ChosenCand}(p)$	$\to$	$p.choice \leftarrow \texttt{Choice}(p)$ $p.nbA \leftarrow \texttt{NbA}(p)$ $p.busy \leftarrow \texttt{IsBusy}(p)$
`Count`	::	$p.nbA \neq \texttt{NbA}(p)$	$\to$	$p.nbA \leftarrow \texttt{NbA}(p)$
`Flag`	::	$p.busy \neq \texttt{IsBusy}(p)$	$\to$	$p.busy \leftarrow \texttt{IsBusy}(p)$
`Leave`	::	$\texttt{CanLeave}(p)$	$\to$	$p.inA \leftarrow false$

Hence, to leave A, a process p should not authorize any neighbor to leave A ($p.choice = \bot$) and should be authorized to leave by all of its neighbors ($\forall q \in \mathcal{N}_p, q.choice = p$). For example, consider the $(1, 0)$-alliance in Figure 1. Only Process 2 is able to leave A. Now, Process 2 can actually leave A because it has enough neighbors in A (*i.e.*, 2 neighbors, while $f(2) = 1$). So, if it leaves A, then it will still have two neighbors in A: Requirement 1 will be not violated.

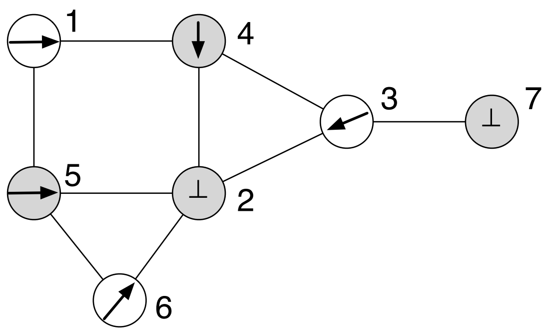

Figure 1: Neighbor pointers in a $(1, 0)$-alliance. Numbers indicate IDs; the set of gray nodes represents A. Arrows designate the neighbor pointed by the node. "$\bot$" inside a node indicates that the node designates no neighbor.

Ensuring Requirement 2. This requirement is also maintained by the fact that a process p must have an authorization from each of its neighbors q before leaving A. A neighbor q can give such an authorization to p only if q still has enough neighbors in A without p. For a process q to authorize a neighbor q' to leave A, q' must currently be in A, *i.e.*, $q'.inA = true$, and q must have more neighbors than necessary in A, *i.e.*, the predicate $\texttt{HasExtra}(q)$ should be true, meaning that $\mathcal{N}_q \cap A$ has more than $g(q)$, respectively $f(q)$, members if q is in A, respectively not in A. For example, consider the $(1, 0)$-alliance in Figure 1. Processes 4 and 5 can designate Process 2 because they belong to A and $g(4) = g(5) = 0$. Moreover, Processes 3 and 6 can designate Process 2 because they do not belong to A and $f(3) = f(6) = 1$: if Process 2 leaves A, Process 3 (resp. Process 6) still has one neighbor in A, which is Process 7 (resp. Process 5).

Busy Processes. It is possible that a neighbor q' of q cannot leave A — in this case q' is said to be *busy* — because one of these two conditions is *true*:

(i) $\texttt{NbA}(q') < f(q')$**:** in this case, q' does not have enough neighbors in A to be allowed to leave it.

(ii) $\neg\texttt{IsExtra}(q')$**:** in this case, at least one neighbor of q' needs q' to stay in A.

If q chooses such a neighbor q', this may lead to a deadlock. We use the Boolean variable $q'.busy$ to inform q that one of the two aforementioned conditions holds for q'. Action `Flag` maintains $q'.busy$. So, to prevent deadlock, q must not choose any neighbor q' for which $q'.busy = true$.

q' evaluates Condition (i) by reading the variables *inA* of all its neighbors. On the other hand, Condition (ii) requires that q' knows for each of its neighbors, both their status (*inA*) and the number of their own neighbors that are in A. This latter information is obtained using an additional variable, *nbA*, where each process maintains, using Action `Count`, the number of its neighbors that are in A.

Consider the $(2, 0)$-alliance in Figure 2. Process 5 is busy because of Condition (i): it has only one neighbor in A, while $f(5) = 2$. Process 2 is busy because of Condition (ii): its neighbor 1 is not in A, $f(1) = 2$, and has only 2 neighbors in A, so it cannot authorize any of its neighbors to leave. Consequently, Process 1 cannot designate any neighbor (all its neighbors in A are busy); while Process 3 should not designate Process 2.

Action `Vote`. Hence, the value of *p.choice* is chosen, using Action `Vote`, as follows:

1. *p.choice* is set to $\bot$ if the condition $\texttt{Cand}(p) \neq \emptyset \wedge \texttt{HasExtra}(p) \wedge (\texttt{IamCand}(p) \Rightarrow \texttt{MinCand}(p) < p)$ in Macro $\texttt{ChosenCand}(p)$ is *false*, *i.e.*, if one of the following conditions holds:

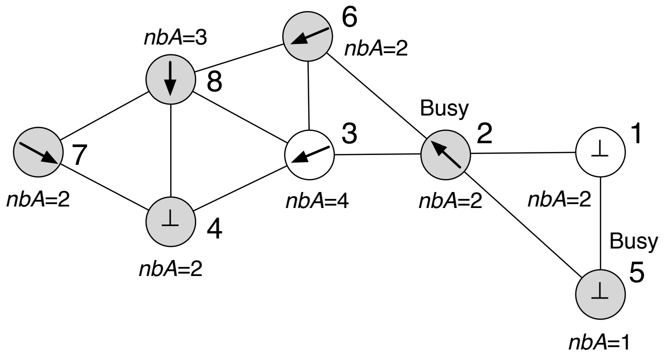

Figure 2: Busy processes in a $(2,0)$-alliance. Busy processes are indicated; Value of *nbA* is also given.

- $\texttt{Cand}(p) = \emptyset$, which means that no neighbor of p can leave A.
- $\texttt{HasExtra}(p) = false$, which means that p cannot authorize any neighbor to leave A.
- $\texttt{IamCand}(p) \wedge p < \texttt{MinCand}(p)$, which means that p is also candidate to leave A and has higher priority to leave A than any other candidate in its neighborhood. (Remember that to be allowed to leave A, p should, in particular, satisfy $p.choice = \perp$.)

  The aforementioned priorities are based on process IDs, *i.e.*, for every two process u and v, u has higher priority than v if and only if the ID of u is smaller than the ID of v.

2. Otherwise, p uses $p.choice$ to designate a neighbor that is in A and not busy in order to authorize it to leave A. If p has several possible candidates among its neighbors, it selects the one of highest priority (*i.e.*, of smallest ID). For example, if we consider the $(2,0)$-alliance in Figure 2, then we can see that Process 3 designates Process 4 because it is its smallest neighbor that is both in A and not busy.

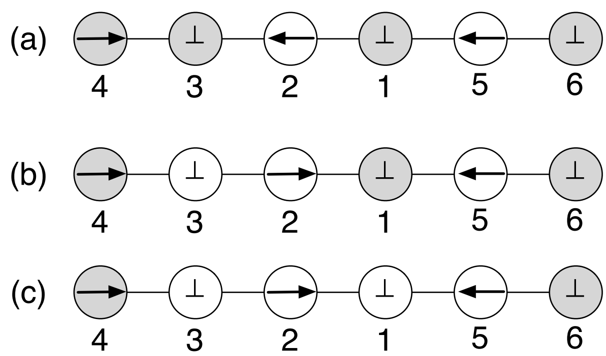

Figure 3: Requirement 2 violation in a $(1,0)$-alliance. We only show values that are useful in the reasoning.

There is one last problem: A process q may change its pointer while simultaneously one of its neighbors q' leaves A, and consequently Requirement 2 may be violated. Indeed, q chooses new candidate assuming that q' remains in A. This may happens only if the previous value of $q.choice$ was q'. To avoid this situation, we do not allow q to directly change $q.choice$ from one neighbor to another. Each time q wants to change its pointer, if $q.choice \in \mathcal{N}_q$, q first resets $q.choice$ to $\perp$, see $\texttt{Choice}(q)$.

Figures 3 and 4 illustrates this last issue in the case of a $(1,0)$-alliance. In the step from Configuration (a) to Configuration (b) of Figure 3, Process 2 directly changes its pointer from 3 to 1. Now, simultaneously, 3 leaves A. So, Process 2 authorizes Process 1 to leave A, while it should not do. Now, after that, Process 1 is authorized to leave A and does it in Step from Configuration (b) to Configuration (c):

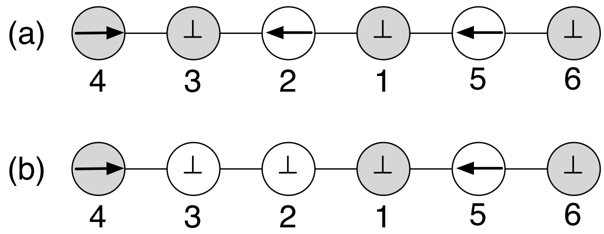

Figure 4: The reset of the neighbor pointer is applied to the example of Figure 3 ($(1, 0)$-alliance).

Requirement 2 is violated. Figure 4 illustrates how we solve the problem. In Configuration (b), Process 3 has left, but the pointer of Process 2 is equal to $\bot$. So, Process 1 cannot leave yet and by the way, Process 2 will not authorize it to leave.

3.2 Joining A

Action `Join` allows a process to join A. A process p not in A must join A if:

(1) p has not enough neighbors in A ($\texttt{NbA}(p) < f(p)$), or

(2) a neighbor of p needs p to join A ($\texttt{IsMissing}(p)$).

Moreover, to prevent p from cycling in and out of A, we require that every neighbor of p stops designating it (with their *choice* pointer) before p can join A (again). Note that all neighbors of p stop designating p immediately after it leaves A, see Action `Vote`. (Actually, this introduces a delay of only one round.)

A process evaluates condition (1) by reading the variables *inA* of all its neighbors. To evaluate condition (2), it needs to know for each neighbor q, both its status *w.r.t.* A ($q.inA$) and the number of its neighbors that are in A ($q.nbA$).

4 Correctness

Recall that in any configuration γ, we define the set $A_\gamma = \{p \in V, p.inA\}$. (We omit the subscript γ when it is clear from the context.) In the next subsection, we define some predicates. Subsection 4.2 is dedicated to the proof of self-stabilization of $\mathcal{MA}(f, g)$ assuming an unfair daemon. We study the safe convergence of $\mathcal{MA}(f, g)$ in Subsection 4.3.

4.1 Predicates

First, throughout the section, we will use the notion of a *closed predicate*: Let P be a predicate over configuration of $\mathcal{MA}(f, g)$. P is *closed* if and only if $\forall \gamma, \gamma' \in \mathcal{C}$, $P(\gamma) \wedge \gamma \mapsto \gamma' \Rightarrow P(\gamma')$.

Let now define some predicates. First, for every process p,

$$\texttt{Fga}(p) \stackrel{\text{def}}{=} (\neg p.inA \Rightarrow \texttt{NbA}(p) \geq f(p)) \wedge (p.inA \Rightarrow \texttt{NbA}(p) \geq g(p))$$

When a process p satisfies $\texttt{Fga}(p)$, this means that it is locally correct, *i.e.*, it has enough neighbors in A according to its status. Then, by definition we have:

Remark 1 *A is an (f, g)-alliance if and only if $\forall p \in V$, $\texttt{Fga}(p)$.*

For every process p,

$$\texttt{NbAOk}(p) \stackrel{\text{def}}{=} (\neg p.inA \Rightarrow p.nbA \geq f(p)) \wedge (p.inA \Rightarrow p.nbA \geq g(p))$$

This predicate is always used in conjunction with $\texttt{Fga}(p)$. When both predicates are *true* at p, this means that p is locally correct and the variable $p.nbA$ gives this information to the neighbors of p.

For every process p,

$$\texttt{ChoiceOk}(p) \stackrel{\text{def}}{=} (p.choice \neq \bot \wedge p.choice.inA) \Rightarrow \texttt{HasExtra}(p)$$

Once $\texttt{ChoiceOk}(p)$ holds at p, no neighbor of p can make p locally incorrect by leaving A.

The following predicates are defined over configurations of $\mathcal{MA}(f,g)$:

$$SP_{1-Minimal} \stackrel{\text{def}}{=} A \text{ is a 1-minimal } (f,g)\text{-alliance}$$
$$SP_{Minimal} \stackrel{\text{def}}{=} A \text{ is a minimal } (f,g)\text{-alliance}$$

4.2 Self-stabilization of $\mathcal{MA}(f,g)$

Partial Correctness. We now show that in any terminal configuration γ, the specification of $\mathcal{MA}(f,g)$ is achieved. To see this, we first show that A is an (f,g)-alliance in γ (Lemma 2), then we show that A is 1-minimal in γ, so if $f \geq g$, A is also a minimal (f,g)-alliance (Lemma 3). To show these two results, we use two intermediate claims: Lemma 1 and Corollary 1. The former states that every process of A is busy in γ, meaning that either p has not enough neighbors in A to leave A, or at least one neighbor of p requires that p stays in A, *i.e.,* A is 1-minimal. The latter is a simple corollary of Lemma 1 and states that no process authorizes a neighbor to leave A in γ.

In any terminal configuration, Action $\texttt{Count}$ is disabled at every process, so:

Remark 2 *In any terminal configuration of $\mathcal{MA}(f,g)$, for every process p, p.nbA$=\texttt{NbA}(p)=|\{q \in \mathcal{N}_p, q.\text{inA}\}|$.*

Lemma 1 *In any terminal configuration of $\mathcal{MA}(f,g)$, for every process p, p.inA $\Rightarrow$ p.busy.*

Proof. By contradiction. Let γ be a terminal configuration of $\mathcal{MA}(f,g)$ and assume that there is at least one process p such that $p.inA = true$ and $p.busy = false$ in γ. Then, for each such process p, we have $\texttt{IsBusy}(p) = false$ in γ, because Action $\texttt{Flag}$ is disabled at every process.

Let

$$p_{\min} = \min\{p \in V, p.inA = true \wedge p.busy = false\} \text{ in } \gamma \qquad (1)$$

Since $\neg\texttt{IsBusy}(p_{\min})$ in γ, we also have:

$$\begin{array}{ll} \texttt{IsExtra}(p_{\min}) & \\ \forall q \in \mathcal{N}_{p_{\min}}, (\neg q.inA \Rightarrow q.nbA > f(q)) \wedge (q.inA \Rightarrow q.nbA > g(q)) & \\ \forall q \in \mathcal{N}_{p_{\min}}, (\neg q.inA \Rightarrow \texttt{NbA}(q) > f(q)) \wedge (q.inA \Rightarrow \texttt{NbA}(q) > g(q)) & \text{by Remark 2} \\ \forall q \in \mathcal{N}_{p_{\min}}, \texttt{HasExtra}(q) & (2) \end{array}$$

Then, because $p_{\min}.inA = true \wedge p_{\min}.busy = false$ in γ we have:

$$\begin{array}{ll} \forall q \in \mathcal{N}_{p_{\min}}, p_{\min} \in \texttt{Cand}(q) & (3) \\ \forall q \in \mathcal{N}_{p_{\min}}, \texttt{Cand}(q) \neq \emptyset & (4) \end{array}$$

By (1) and (3), in γ we have:

$$\forall q \in \mathcal{N}_{p_{\min}}, \texttt{MinCand}(q) = p_{\min} \qquad (5)$$

By (1) and (5), in γ we have:

$$\forall q \in \mathcal{N}_{p_{\min}}, (\texttt{IamCand}(q) \Rightarrow \texttt{MinCand}(q) < q) \qquad (6)$$

By (2), (4), (5), (6) and the fact that Action $\texttt{Vote}$ is disabled, in γ we have:

$$\begin{array}{ll} \forall q \in \mathcal{N}_{p_{\min}}, \texttt{ChosenCand}(q) = p_{\min} & \\ \forall q \in \mathcal{N}_{p_{\min}}, q.choice = p_{\min} & (7) \end{array}$$

By definition, $\texttt{IamCand}(p_{\min})$ holds in γ. Moreover, by (1), $\texttt{MinCand}(p_{\min}) > p_{\min}$ in γ. So, $\texttt{MinCand}(p_{\min}) < p_{\min}$ is *false* in γ. Hence, in γ we have $(\texttt{IamCand}(p_{\min}) \Rightarrow \texttt{MinCand}(p_{\min}) < p_{\min}) = \mathit{false}$, and consequently:

$$\begin{array}{ll} \texttt{ChosenCand}(p_{\min}) = \bot & \\ p_{\min}.\mathit{choice} = \bot & \text{(Action Vote is disabled)} \quad (8) \end{array}$$

Finally, because $\neg\texttt{IsBusy}(p_{\min})$ holds in γ, we have $\texttt{NbA}(p_{\min}) \geq f(p_{\min})$ in γ. So, by (7), (8), and the fact that $p_{\min}.\mathit{inA} = \mathit{true}$ in γ, we can conclude that $\texttt{CanLeave}(p_{\min})$ holds in γ, that is, $p_{\min}$ is enabled in γ, contradiction. □

By Lemma 1, for every process p, $\texttt{Cand}(p) = \emptyset$ in any terminal configuration γ. Thus $\texttt{ChosenCand}(p) = \bot$ in γ, and from the negation of the guard of Action Vote, we have:

Corollary 1 *In any terminal configuration of $\mathcal{MA}(f,g)$, for every process p, p.choice $= \bot$.*

Lemma 2 *In any terminal configuration of $\mathcal{MA}(f,g)$, A is an (f,g)-alliance.*

Proof. Let γ be a terminal configuration. By Remark 1, we merely need show that every process p satisfies $\texttt{Fga}(p)$ in γ. Consider the following two cases:

$p \notin A$ in γ: First, by definition, $p.\mathit{inA} = \mathit{false}$ in γ. Then, γ being terminal, $\neg\texttt{MustJoin}(p)$ holds in γ. $\neg\texttt{MustJoin}(p) = \neg(\neg p.\mathit{inA} \wedge (\texttt{NbA}(p) < f(p) \vee \texttt{IsMissing}(p)) \wedge (\forall q \in \mathcal{N}_p, q.\mathit{choice} \neq p)) = p.\mathit{inA} \vee (\texttt{NbA}(p) \geq f(p) \wedge \neg\texttt{IsMissing}(p)) \vee (\exists q \in \mathcal{N}_p, q.\mathit{choice} = p)$. By $p.\mathit{inA} = \mathit{false}$ and Corollary 1, $\neg\texttt{MustJoin}(p)$ in γ implies that $\texttt{NbA}(p) \geq f(p) \wedge \neg\texttt{IsMissing}(p)$ in γ. So, $\neg p.\mathit{inA} \wedge \texttt{NbA}(p) \geq f(p)$ holds in γ, which implies that $\texttt{Fga}(p)$ holds in γ.

$p \in A$ in γ: First, by definition, $p.\mathit{inA} = \mathit{true}$ in γ. We need to show that $\texttt{Fga}(p) = \mathit{true}$ in γ. Assume $\texttt{Fga}(p) = \mathit{false}$. Then, $\texttt{NbA}(p) < g(p)$. As $\delta_p \geq g(p)$, $\exists q \in \mathcal{N}_p$, $\neg q.\mathit{inA}$ in γ. By Remark 2, $p.\mathit{nbA} < g(p)$ in γ. So, as $p \in \mathcal{N}_q$, $\texttt{IsMissing}(q)$ holds in γ. Now, as $q.\mathit{inA} = \mathit{false}$ and $\texttt{IsMissing}(q) = \mathit{true}$ in γ, by Corollary 1, we can conclude that $\texttt{MustJoin}(q)$ holds in γ, that is, q is enabled in γ, contradiction.

□

Lemma 3 *In any terminal configuration of $\mathcal{MA}(f,g)$, A is a 1-minimal (f,g)-alliance, and if $f \geq g$, then A is a minimal (f,g)-alliance.*

Proof. Let γ be a terminal configuration. We already know that in γ, A defines an (f,g)-alliance. Moreover, by Property 1, if A is 1-minimal and $f \geq g$, then A is a minimal (f,g)-alliance. Thus, we only need to show the 1-minimality of A.

Assume that A is not 1-minimal. Then there is a process $p \in A$ such that $A - \{p\}$ is an (f,g)-alliance. So:

1. $|A \cap \mathcal{N}_p| \geq f(p)$,
2. $\forall q \in \mathcal{N}_p, q \in A \Rightarrow |A \cap \mathcal{N}_q - \{p\}| \geq g(q)$, and
3. $\forall q \in \mathcal{N}_p, q \notin A \Rightarrow |A \cap \mathcal{N}_q - \{p\}| \geq f(q)$.

By 1, in γ we have:

$$\texttt{NbA}(p) \geq f(p) \qquad (a)$$

By 2, in γ we have:

$$\begin{array}{lll} \forall q \in \mathcal{N}_p, q.\mathit{inA} \Rightarrow \texttt{NbA}(q) - 1 \geq g(q) & & \\ \forall q \in \mathcal{N}_p, q.\mathit{inA} \Rightarrow \texttt{NbA}(q) > g(q) & & \\ \forall q \in \mathcal{N}_p, q.\mathit{inA} \Rightarrow q.\mathit{nbA} > g(q) & \text{by Remark 2} & (b) \end{array}$$

By 3, in γ we have:

$$\forall q \in \mathcal{N}_p, \neg q.inA \Rightarrow \texttt{NbA}(q) - 1 \geq f(q)$$
$$\forall q \in \mathcal{N}_p, \neg q.inA \Rightarrow \texttt{NbA}(q) > f(q)$$
$$\forall q \in \mathcal{N}_p, \neg q.inA \Rightarrow q.nbA > f(q) \qquad \text{by Remark 2} \qquad (c)$$

By (b) and (c), $\texttt{IsExtra}(p)$ holds in γ. So, by (a), $\texttt{NbA}(p) \geq f(p) \wedge \texttt{IsExtra}(p)$ holds in γ, that is, $\neg\texttt{IsBusy}(p)$ holds in γ. Now, `Flag` is disabled at p in γ , so $p.busy = \mathit{false}$ in γ. As we assumed that $p.inA = \mathit{true}$ in γ ($p \in A$), this contradicts Lemma 1. □

Termination. We now show that, if $f \geq g$, the unfair daemon cannot prevent $\mathcal{MA}(f,g)$ from terminating, starting from any configuration. The proof consists in showing that the number of steps to reach a terminal configuration, starting from any arbitrary configuration, is bounded, no matter the choices of daemon are.

Let J be the maximum number of times any process executes Action `Join` in any execution. Lemma 4, below, states that the number of steps to reach a terminal configuration of $\mathcal{MA}(f,g)$ depends on J, as well as on both global parameters of the network, its degree Δ, and its size n.

Lemma 4 *Starting from any configuration, $\mathcal{MA}(f,g)$ reaches a terminal configuration in $O(J\Delta^3 n)$ steps.*

Proof. Consider any process p in any execution e of $\mathcal{MA}(f,g)$. Let $J(p)$, $L(p)$, $C(p)$, $F(p)$, and $V(p)$ be the number of times p executes Actions `Join`, `Leave`, `Count`, `Flag` and `Vote` in e, respectively. By definition, $J(p) \leq J$.

After executing `Leave`, p should execute `Join` before executing `Leave` again. So:

$$L(p) \leq 1 + J(p) \leq 1 + J$$

In the following, we use the number of times p modifies the value of its variable $p.nbA$. This number is denoted by $\sharp nbA(p)$. $p.nbA$ is modified because either $p.nbA \neq \texttt{NbA}(p)$ in the initial configuration, or $p.nbA \neq \texttt{NbA}(p)$ becomes *true* after a neighbor of p joins or leaves A. So:

$$\sharp nbA(p) \leq 1 + \sum_{q \in \mathcal{N}_p} (J(q) + L(q)) \leq 1 + \Delta(2J+1)$$

By definition, p executes Action `Count` at most $\sharp nbA(p)$ times. So:

$$C(p) \leq \sharp nbA(p) \leq 1 + \Delta(2J+1)$$

In the following, we use the number of times p modifies the value of its variable $p.busy$. This number is denoted by $\sharp busy(p)$. $p.busy$ is modified because either $p.busy \neq \texttt{IsBusy}(p)$ holds in the initial configuration, or $p.busy \neq \texttt{IsBusy}(p)$ becomes *true* after a neighbor q of p joins or leaves A, or modifies its counter $q.nbA$. So:

$$\sharp busy(p) \leq 1 + \sum_{q \in \mathcal{N}_p} (J(q) + L(q) + \sharp nbA(q)) \leq 1 + (2+2J)\Delta + (1+2J)\Delta^2$$

By definition, p executes Action `Flag` at most $\sharp busy(p)$ times. So:

$$F(p) \leq \sharp busy(p) \leq 1 + (2+2J)\Delta + (1+2J)\Delta^2$$

Action `Vote` is enabled when p wants to change its pointer $p.choice$. That is, either (1) p does not want to authorize any neighbor to leave A (in this case, its pointer is reset to $\perp$), or (2) p has a new favorite candidate. In the latter case, p may be required to reset its pointer to $\perp$ first, because we impose a strict alternation in $p.choice$ between values of $\mathcal{N}_p$ and $\perp$. Hence, p may require up to two executions of Action `Vote` to fix the value of $p.choice$.

As for other actions, `Vote` can be initially enabled. Moreover, either case (1) or (2) occurs for p every time either (i): the variables *inA* of p or its neighbors are modified, or (ii): the variable *busy* or *nbA* of one or more of its neighbors is modified. Therefore

$$V(p) \leq 2(1+\sum_{r\in\mathcal{N}_p\cup\{p\}}(J(r)+L(r))+\sum_{q\in\mathcal{N}_p}(\sharp busy(q)+\sharp nbA(q)))$$
$$V(p) \leq 4+4J+\Delta(6+4J)+\Delta^2(6+8J)+\Delta^3(2+4J)$$

So, the maximum number of steps before $\mathcal{MA}(f,g)$ reaches a terminal configuration is:

$$n(J(p)+L(p)+C(p)+F(p)+V(p)) \leq n[7+6J+\Delta(9+8J)+\Delta^2(7+10J)+\Delta^3(2+4J)] = O(J.\Delta^3.n)$$

□

To complete the proof of convergence of $\mathcal{MA}(f,g)$, we now show, in Lemma 11, that J is bounded by 1 if $f \geq g$. This lemma uses six technical results, given in Lemmas 5 through 10.

Lemma 5 *Let p be a process. $\forall q, q' \in \mathcal{N}_p \cup \{p\}$, if $q' \neq q$, then q and q' cannot leave A in the same step.*

Proof. By contradiction. Assume, that there are two processes $q, q' \in \mathcal{N}_p \cup \{p\}$ such that $q' \neq q$, and both q and q' leave the alliance in some step $\gamma \mapsto \gamma'$. Consider the two following cases:

- $q = p \vee q' = p$**:** Without loss of generality, assume that $q' = p$. From the guard of Action `Leave` at p, $p.choice = \bot$. Now, $p \in \mathcal{N}_q$, so from the guard of Action `Leave` at q, $p.choice = q \neq \bot$, a contradiction.
- $q \neq p \wedge q' \neq p$**:** By definition, $p \in \mathcal{N}_q$ and $p \in \mathcal{N}_{q'}$. So, from the guard of Action `Leave` at q, we have $p.choice = q$; and from the guard of Action `Leave` at q', $p.choice = q'$, a contradiction.

□

Corollary 2 *If a process p leaves A in the step $\gamma \mapsto \gamma'$, then $\texttt{Fga}(p)$ holds in γ'.*

Proof. Assume that process p leaves A in $\gamma \mapsto \gamma'$. From the guard of Action `Leave`, we have $\texttt{NbA}(p) \geq f(p)$. By Lemma 5, no neighbor of p leaves A in $\gamma \mapsto \gamma'$. So, $p.inA = false$ and $\texttt{NbA}(p) \geq f(p)$ in γ', and we are done. □

Lemma 6 *If a process p executes `Leave` or p.choice is assigned the ID of some neighboring process in $\gamma \mapsto \gamma'$, then $\texttt{NbAOk}(p)$ holds in γ'.*

Proof. Let X be the value of $\texttt{NbA}(p)$ in γ.

If p executes `Leave` in $\gamma \mapsto \gamma'$, then from the guard of `Leave`, we know that $X \geq f(p)$. Moreover, as Action `Count` is disabled at p (otherwise, `Leave` is not executed because `Count` has higher priority), $p.nbA = X$ in γ. So, $p.inA = false$ and $p.nbA = X \geq f(p)$ in γ', *i.e.*, $\texttt{NbAOk}(p)$ holds in γ'.

If p executes $p.choice \leftarrow q \in \mathcal{N}_p$ in $\gamma \mapsto \gamma'$, then $\texttt{HasExtra}(p)$ holds in γ, p does not change the value of $p.inA$ in $\gamma \mapsto \gamma'$, and $p.nbA \leftarrow X$ in $\gamma \mapsto \gamma'$. Consequently, $\texttt{NbAOk}(p)$ holds in γ'. □

Lemma 7 *For every process p, $\texttt{ChoiceOk}(p)$ is closed.*

Proof. By contradiction. Assume that there is a process p such that $\texttt{ChoiceOk}(p)$ is not closed: There exists a step $\gamma_i \mapsto \gamma_{i+1}$ where $\texttt{ChoiceOk}(p)$ holds in γ_i, but not in γ_{i+1}. That is: $p.choice \neq \bot \wedge p.choice.inA \wedge \neg\texttt{HasExtra}(p)$ holds in γ_{i+1}.

Assume that the value of $p.inA$ changes between γ_i and γ_{i+1}. Then, p executes `Join` or `Leave` in $\gamma_i \mapsto \gamma_{i+1}$. In the former case, $p.choice = \bot$ in γ_{i+1}, and consequently, $\texttt{ChoiceOk}(p)$ still holds in γ_{i+1}, contradiction. In the latter case, from the guard of `Leave`, we can deduce that $p.choice = \bot$ in γ_i and, as Action `Leave` does not modify the variable *choice*, $p.choice = \bot$ still holds in γ_{i+1}, contradiction. So, the value of $p.inA$ does not change during $\gamma_i \mapsto \gamma_{i+1}$. Consider the following two cases:

A) $p.choice = \bot$ in γ_i: $p.choice \neq \bot$ in γ_{i+1}. So, p executes Action `Vote` in $\gamma_i \mapsto \gamma_{i+1}$. Consequently, the guard of Action `Vote` holds at p in γ_i. In particular, $\texttt{ChosenCand}(p) \neq \bot$ in γ_i, and so $\texttt{HasExtra}(p)$ also holds in γ_i. As the value of $p.inA$ does not change during $\gamma_i \mapsto \gamma_{i+1}$, a neighbor of p should leave A during $\gamma_i \mapsto \gamma_{i+1}$, so that $\texttt{HasExtra}(p)$ becomes *false*. Since $p.choice = \bot$ in γ_i, no neighbor of p can execute Action `Leave` in $\gamma_i \mapsto \gamma_{i+1}$, contradiction.

B) $p.choice \neq \bot$ in γ_i: If p executes `Vote` in $\gamma_i \mapsto \gamma_{i+1}$, then $p.choice = \bot$ in γ_{i+1} and `ChoiceOk`(p) still holds in γ_{i+1}, contradiction. So, the value of $p.choice$ is the same in γ_i and γ_{i+1}. Let q be this value. Recall that $q \in \mathcal{N}_p$, and consider the following two subcases:

$\neg q.inA$ in γ_i: $q.inA$ holds in γ_{i+1}. So, q executes Action `Join` in $\gamma_i \mapsto \gamma_{i+1}$. Now, as $p.choice = q$ in γ_i, Action `Join` is disabled at q in γ_i, contradiction.

$q.inA$ in γ_i: Since `ChoiceOk`(p) holds in γ_i, we have `HasExtra`(p) = *true* in γ_i. Now, `HasExtra`(p) is *false* in γ_{i+1}. Moreover, we already know that the value of $p.inA$ does not change during $\gamma_i \mapsto \gamma_{i+1}$. So, by Lemma 5, exactly one neighbor of p executes Action `Leave` in $\gamma_i \mapsto \gamma_{i+1}$. As $p.choice = q$ in γ_i, the neighbor that leaves A in $\gamma_i \mapsto \gamma_{i+1}$ is necessarily q. So, $q.inA$ = *false* in γ_{i+1}, and since $p.choice = q$ still holds in γ_{i+1}, we have $p.choice.inA$ = *false* in γ_{i+1}. Consequently, `ChoiceOk`(p) still holds in γ_{i+1}, contradiction.

□

Lemma 8 *For every process p,* `ChoiceOk`(p) *holds forever after p executes any action.*

Proof. Let p be a process that executes any action in $\gamma \mapsto \gamma'$. By Lemma 7, we only need to show that `ChoiceOk`(p) is *true* in either γ or γ'.

Consider the following three cases:

A) p executes `Join`: Then, $p.choice = \bot$ in γ', and consequently `ChoiceOk`(p) is *true* in γ'.

B) p executes `Vote`: Then, $p.choice = \bot$ in either γ or γ', and `ChoiceOk`(p) is *true* in γ or γ'.

C) p executes any other action: As in the previous cases, if $p.choice = \bot$ in γ, we conclude that `ChoiceOk`(p) is *true* in γ. Suppose $p.choice \neq \bot$ in γ. Since `Join` and `Vote` have higher priority than any other action, we deduce that their respective guards are *false* in γ. In particular, from the negation of the guard of Action `Vote`, we can deduce that $p.choice$ = `ChosenCand`$(p) \neq \bot$ in γ. So, `HasExtra`(p) holds in γ, and thus `ChoiceOk`(p) holds in γ. □

Lemma 9 *If $f \geq g$,* `ChoiceOk`$(p) \wedge$ `Fga`(p) *is closed for every process p.*

Proof. Let p be a process. Let $\gamma \mapsto \gamma'$ be any step such that `ChoiceOk`$(p) \wedge$ `Fga`(p) holds in γ. By Lemma 7, we have: *(*)* `ChoiceOk`(p) *holds in* γ'.

Hence, we only need to show that `Fga`(p) still holds in γ'. Let X be the value of `NbA`(p) in γ. Let Y be the value of `NbA`(p) in γ'. By Lemma 5, $Y \geq X - 1$. Consider the following two cases:

- *A) The value of p.inA is the same in γ and γ'.*

  If $p.choice = \bot$ in γ, then no neighbor of p can leave A in $\gamma \mapsto \gamma'$. Consequently, $Y \geq X$, which also implies that `Fga`(p) still holds in γ'.

  Otherwise, $p.choice \neq \bot$ in γ. There are two cases.

  $p.choice.inA$ in γ: By (*), $p.inA \Rightarrow X > g(p)$ and $\neg p.inA \Rightarrow X > f(p)$ in γ. So, as the value of $p.inA$ is the same in γ and γ', and $Y \geq X - 1$, we have $p.inA \Rightarrow Y \geq g(p)$ and $\neg p.inA \Rightarrow Y \geq f(p)$ in γ', which implies that `Fga`(p) still holds in γ'.

  $\neg p.choice.inA$ in γ: There is no neighbor q of p such that $q.inA$ and $p.choice = q$ in γ. So, no neighbor of p leaves A in $\gamma \mapsto \gamma'$. Consequently, $Y \geq X$ and, as the value of $p.inA$ is the same in γ and γ', `Fga`(p) still holds in γ'.

- *B) p changes the value of p.inA in $\gamma \mapsto \gamma'$.* Consider the following two cases:

  p executes `Leave` in $\gamma \mapsto \gamma'$: First, $p.inA$ = *false* in γ'. So, `Fga`(p) holds in γ' only if $Y \geq f(p)$. Then, from the guard of Action `Leave`, we have (1) $X \geq f(p)$ and (2) $p.choice = \bot$ in γ. By (2), no neighbor of p leaves A in $\gamma \mapsto \gamma'$. So, $Y \geq X \geq f(p)$, which implies that `Fga`(p) still holds in γ'.

  p executes `Join` in $\gamma \mapsto \gamma'$: First, $p.inA$ = *true* in γ'. So, `Fga`(p) holds in γ' only if $Y \geq g(p)$. (Recall that $f(p) \geq g(p)$.) Consider the following two cases:

$X > Y$: Then $Y = X - 1$. Let q be the neighbor of p that leaves A in $\gamma \mapsto \gamma'$. $q.inA = true \wedge p.choice = q$ in γ. So, by (*), $p.inA = false$ in γ implies that $X > f(p)$. So, $Y \geq f(p) \geq g(p)$, which implies that $\texttt{Fga}(p)$ still holds in γ'.

$X \leq Y$: Then, $Y \geq X \geq f(p) \geq g(p)$, which implies that $\texttt{Fga}(p)$ still holds in γ'.

□

Lemma 10 *Assuming $f \geq g$, we have: for every process p, $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ is closed.*

Proof. Let p be a process. Let $\gamma \mapsto \gamma'$ be any step such that $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ holds in γ. By Lemma 9, $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ is *true* in γ'. So, we only need to show that $\texttt{NbAOk}(p)$ still holds in γ'.

Assume the contrary. Let X be the value of $\texttt{NbA}(p)$ in γ and consider the following two cases:

- *p does not change the value of p.inA in $\gamma \mapsto \gamma'$.* Assume that $p.inA$ is *true* in γ. Then, p must modify $p.nbA$ in $\gamma \mapsto \gamma'$ to violate $\texttt{NbAOk}(p)$ in γ'. From the algorithm, p executes $p.nbA \leftarrow X$ in $\gamma \mapsto \gamma'$. Then, $X \geq g(p)$ since $\texttt{Fga}(p)$ in γ. Thus, $p.inA = true$ and $p.nbA \geq g(p)$ in γ', *i.e.*, $\texttt{NbAOk}(p)$ still holds in γ', contradiction.

  Assume that $p.inA$ is *false* in γ. By similar reasoning, we obtain a contradiction in this case as well.

- *p changes the value of p.inA in $\gamma \mapsto \gamma'$.* There are two cases:

  p leaves A in $\gamma \mapsto \gamma'$: Then, $\texttt{NbAOk}(p)$ still holds in γ' by Lemma 6, contradiction.

  p joins A in $\gamma \mapsto \gamma'$: Then, $X \geq f(p)$ because $p.inA = false$ and $\texttt{Fga}(p)$ holds in γ. Then, $p.nbA \leftarrow X$ in $\gamma \mapsto \gamma'$. So, $p.inA = true$ and $p.nbA \geq f(p) \geq g(p)$ in γ', *i.e.*, $\texttt{NbAOk}(p)$ still holds in γ', contradiction.

□

Lemma 11 *If $f \geq g$, then in any execution of $\mathcal{MA}(f,g)$, $J \leq 1$, that is, every process joins the (f,g)-alliance at most once.*

(Figure 5 illustrates the following proof.)
Proof. By contradiction. Assume that some process p executes Action `Join` at least two times. Note that p must execute Action `Leave` between two executions of Action `Join`. Thus, there exist $0 \leq i < j < k$ such that p joins A in $\gamma_i \mapsto \gamma_{i+1}$, leaves A in $\gamma_j \mapsto \gamma_{j+1}$, and joins it again in $\gamma_k \mapsto \gamma_{k+1}$.

From the guard of Action `Join`, $q.choice \neq p$ in γ_i for all $q \in \mathcal{N}_p$. From the guard of Action `Leave`, $q.choice = p$ in γ_j for all $q \in \mathcal{N}_p$. Thus:

(1) Every neighbor q of p executes $q.choice \leftarrow p$ using Action `Vote` before γ_j.

Let q be any neighbor of p. Let $\gamma_l \mapsto \gamma_{l+1}$ be a step at which q executes $q.choice \leftarrow p$, using Action `Vote`, for $i < l < j$. Such a step exists by (1). By Lemma 8, $\texttt{ChoiceOk}(q)$ is *true* in γ_{l+1}. Moreover, by (1) and the code of Action `Vote`, we can deduce that (a) $q.choice = \bot$ and (b) $p.inA = true$ in γ_l. By (a), $p.inA$ is still *true* in γ_{l+1}. Now, $q.choice = p$ in γ_{l+1}. So, $\texttt{ChoiceOk}(q)$ in γ_{l+1} implies that $\texttt{HasExtra}(q)$ holds in γ_{l+1}, which in turns implies that $\texttt{Fga}(q)$ holds in γ_{l+1}. Finally, $\texttt{NbAOk}(q)$ in γ_{l+1} by Lemma 6. So, by Lemma 10, $\texttt{ChoiceOk}(q) \wedge \texttt{Fga}(q) \wedge \texttt{NbAOk}(q)$ is true forever from γ_{l+1}. Hence:

(2) Every neighbor q of p satisfies $\texttt{ChoiceOk}(q) \wedge \texttt{Fga}(q) \wedge \texttt{NbAOk}(q)$ forever from γ_j.

As p leaves A in $\gamma_j \mapsto \gamma_{j+1}$, by Corollary 2 and Lemmas 8 and 9, we have:

(3) $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ holds forever from γ_{j+1}.

As p joins A in $\gamma_k \mapsto \gamma_{k+1}$, (a) $\neg p.inA \wedge \texttt{NbA}(p) < f(p)$ or (b) $\texttt{IsMissing}(p)$ holds in γ_k. Now, (a) contradicts (3) and (b) contradicts (2). □

From Lemmas 4 and 11, we deduce the following corollary:

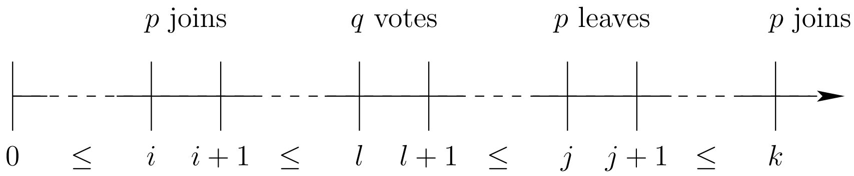

Figure 5: Execution of $\mathcal{MA}(f,g)$

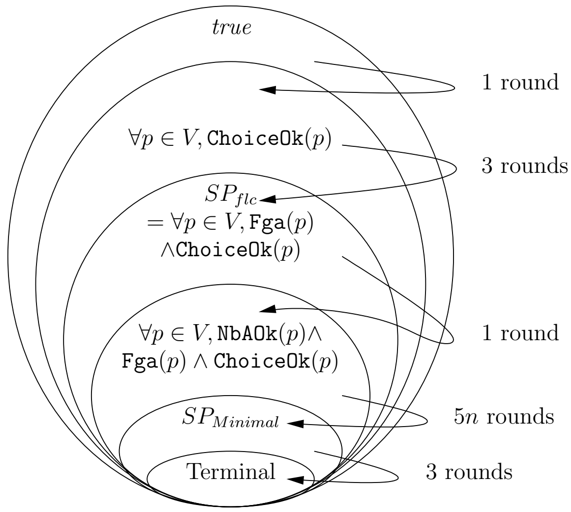

Figure 6: Safe Convergence of $\mathcal{MA}(f,g)$

Corollary 3 *Starting from any configuration, if* $f \geq g$*,* $\mathcal{MA}(f,g)$ *reaches a terminal configuration in* $O(n \times \Delta^3)$ *steps.*

By Lemma 3 and Corollary 3, we have:

Theorem 1 *If* $f \geq g$*,* $\mathcal{MA}(f,g)$ *is silent and self-stabilizing* w.r.t. $SP_{Minimal}$*, and its stabilization time is* $O(\Delta^3 n)$ *steps.*

4.3 Complexity Analysis and Safe Convergence in Rounds

We define a *feasible legitimate configuration* to be any configuration γ that satisfies

$$SP_{flc} \stackrel{\text{def}}{=} \forall p \in V,\ \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$$

In any feasible legitimate configuration, A is an (f,g)-alliance, by Remark 1. Then, from Lemma 9, we already know that the set of *feasible legitimate configurations* is closed if $f \geq g$:

Corollary 4 *If* $f \geq g$*, then* SP_{flc} *is closed.*

To establish safe convergence of $\mathcal{MA}(f,g)$, we show that it gradually converges to more and more specific closed predicates, until reaching a terminal configuration. The gradual convergence to those specific closed predicates is shown in Figure 6.

Lemma 12 *For every process* p*, after at most one round,* $\texttt{ChoiceOk}(p)$ *is true forever.*

Proof. To show this lemma, it is sufficient to show that $\texttt{ChoiceOk}(p)$ becomes *true* during the first round, by Lemma 7. If p is continuously enabled from the initial configuration, then p executes at least one action during the first round and by Lemma 8, we are done.

Otherwise, the first round contains a configuration γ in which every action is disabled at p. In particular, from the negation of the guard of Action Vote, we have $p.choice = \texttt{ChosenCand}(p)$ in γ. Two cases are then possible in γ:

$p.choice = \bot$: In this case, by definition, $\texttt{ChoiceOk}(p)$ holds in γ.

$p.choice \neq \bot$: Then, as $p.choice = \texttt{ChosenCand}(p)$, we have $p.choice = \texttt{MinCand}(p)$ in γ. Thus, $\texttt{HasExtra}(p)$ holds in γ, which implies that $\texttt{ChoiceOk}(p)$ holds in γ.

□

Lemma 13 *Assume $f \geq g$. Let $\gamma_0 \ldots \gamma_i \ldots$ be an execution of $\mathcal{MA}(f,g)$. $\forall i \geq 0$, if $\texttt{ChoiceOk}(p)$ for all $p \in V$ in γ_i, then $\exists j \geq i$ such that γ_j is within at most three rounds from γ_i and $\forall p \in V, \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ holds in γ_j.*

Proof. Let γ_{t_0} be a configuration where $\forall p \in V, \texttt{ChoiceOk}(p)$. Consider any execution (starting in γ_{t_0}) $e = \gamma_{t_0} \ldots \gamma_{t_1} \ldots \gamma_{t_2} \ldots \gamma_{t_3} \ldots$, where $\gamma_{t_1}, \gamma_{t_2}$, and γ_{t_3} are the last configurations of the first, second, and third rounds of e, respectively. By Lemma 7, it is sufficient to show that there is some $t \in [t_0..t_3]$ such that $\forall p \in V, \texttt{Fga}(p)$ in γ_t. Suppose no such a configuration exists. By Lemmas 7 and 9, this means that there exists a process v such that:

(1) $\forall t \in [t_0..t_3], \neg\texttt{Fga}(v)$ in γ_t.

We now derive a contradiction using the following six claims.

(2) $\forall t \in [t_1..t_3], v.choice = \bot$ in γ_t.

Proof of Claim 2: First, by (1), $\forall t \in [t_0..t_3], \neg\texttt{HasExtra}(v)$ in γ_t. So, from the definition $\texttt{ChosenCand}(v)$, we can deduce that $\forall t \in [t_0..t_3]$, if $v.choice = \bot$ in γ_t, then $\forall t' \in [t..t_3], v.choice = \bot$ in $\gamma_{t'}$. Hence, to show the claim, it is sufficient to show that $\exists t \in [t_0..t_1]$ such that $v.choice = \bot$ in γ_t. Suppose the contrary. Then, $\forall t \in [t_0..t_1], v.choice \neq \bot \wedge \neg\texttt{HasExtra}(v)$ in γ_t, that is, the guard of Vote is *true* at v in γ_t. So, v executes (at least) one of the two first actions in the first round to set $v.choice$ to $\bot$, and we are done.

(3) $\forall t \in [t_1..t_3], \neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ in γ_t.

Proof of Claim 3: Let $\gamma_t \mapsto \gamma_{t+1}$ such that $t \in [t_0..t_3 - 1]$. Assume that $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ holds in γ_t.

If $v.inA = true$ in γ_t, then $v.inA = true$ in γ_{t+1} by (1) and Corollary 2, in particular, this implies that $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ still holds in γ_{t+1}. Otherwise, $\neg v.inA \wedge (\forall q \in \mathcal{N}_v, q.choice \neq v)$ holds in γ_t and, from the definition of $\texttt{ChosenCand}(q)$, no neighbor of v can execute Vote to designate v with its pointer during $\gamma_t \mapsto \gamma_{t+1}$. Hence, $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ still holds in γ_{t+1}.

Consequently, $\forall t \in [t_0..t_3]$, if $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ holds in γ_t, then $\forall t' \in [t..t_3]$, $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ still holds in $\gamma_{t'}$. Hence, to show this claim, it is sufficient to show that $\exists t \in [t_0..t_1]$ such that $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ in γ_t. Assume the contrary: $\forall t \in [t_0..t_1]$, $\neg v.inA \wedge (\exists q \in \mathcal{N}_v, q.choice = v)$ holds in γ_t. Then, $\forall q \in \mathcal{N}_v$, if $q.choice \neq v$ in γ_t with $t \in [t_0..t_1]$, then $\forall t' \in [t..t_1], q.choice \neq v$ in $\gamma_{t'}$. So, v has a neighbor q such that $\forall t \in [t_0..t_1], q.choice = v$ in γ_t. Now, in this case, $\forall t \in [t_0..t_1]$, the guard of Vote is *true* at q in γ_t. So, q executes (at least) one of the two first actions in the first round to set $q.choice$ to $\bot$, contradiction.

(4) $\forall t \in [t_2..t_3], v.nbA \leq \texttt{NbA}(v)$ in γ_t.

Proof of Claim 4: First, by (2), no neighbor of v can leave the alliance during the second and third rounds, that is, $\texttt{NbA}(p)$ is monotonically nondecreasing during $[t_1..t_3]$. So, $\forall t \in [t_1..t_3]$, if $v.nbA \leq \texttt{NbA}(v)$ in γ_t, then $\forall t' \in [t..t_3], v.nbA \leq \texttt{NbA}(v)$ in $\gamma_{t'}$. Hence, to show this claim, it is sufficient to show that $\exists t \in [t_1..t_2]$ such that $v.nbA \leq \texttt{NbA}(v)$ in γ_t. Assume the contrary, namely that $v.nbA > \texttt{NbA}(v)$ in γ_t, $\forall t \in [t_1..t_2]$. Then, $\forall t \in [t_1..t_2]$, the guard of Count is *true* at v. Consequently, v executes one of the three first actions, in particular $v.nbA \leftarrow \texttt{NbA}(v)$, during the second round, and, as $\texttt{NbA}(p)$ is monotonically nondecreasing during $[t_1..t_3]$, we obtain a contradiction.

(5) $\forall t \in [t_2..t_3]$, $v.inA$ in γ_t.

Proof of Claim 5: First, $\forall t \in [t_0..t_3]$, if $v.inA = true$ in γ_t, then $\forall t' \in [t..t_3]$, $v.inA = true$ in $\gamma_{t'}$ by (1) and Corollary 2. Hence, to show this claim, it is sufficient to show that $\exists t \in [t_0..t_2]$ such that $v.inA = true$ in γ_t. Assume the contrary: $\forall t \in [t_0..t_2]$, $v.inA = false$ in γ_t. Then, by (1) $\forall t \in [t_0..t_2]$, $\texttt{NbA}(v) < f(v)$ in γ_t. Now, by (3), $\forall t \in [t_1..t_3]$, $\forall q \in \mathcal{N}_v, q.choice \neq v$ in γ_t. So, the guard of the highest priority action of v, `Join`, is true in particular in every configuration γ_t where $t \in [t_1..t_2]$. So, v joins the alliance in the second round, contradiction.

(6) $\forall t \in [t_2..t_3], \forall q \in \mathcal{N}_v, \neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ in γ_t.

Proof of Claim 6: Let q be a neighbor of v. Let $\gamma_t \mapsto \gamma_{t+1}$ such that $t \in [t_1..t_3 - 1]$. Assume that $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds in γ_t.

If $q.inA = true$ in γ_t, then by (2), the guard of `Leave` is disabled at q, so $q.inA = true$ in γ_{t+1}, and consequently, $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ still holds in γ_{t+1}. Otherwise, $\neg q.inA \wedge (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds in γ_t and, from the definition of $\texttt{ChosenCand}(r)$, no neighbor r of q can execute `Vote` to designate q with its pointer during $\gamma_t \mapsto \gamma_{t+1}$. Hence, $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ still holds in γ_{t+1}.

Consequently, $\forall t \in [t_1..t_3]$, $\forall q \in \mathcal{N}_v$, if $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds in γ_t, then $\forall t' \in [t..t_3]$, $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds in $\gamma_{t'}$. Hence, to show this claim, it is sufficient to show that $\forall q \in \mathcal{N}_v$, $\exists t \in [t_1..t_2]$ such that $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ in γ_t. Assume the contrary: let q be a neighbor of v such that $\forall t \in [t_1..t_2]$, $\neg q.inA \wedge (\exists r \in \mathcal{N}_q, r.choice = q)$ holds in γ_t. First, $\forall r \in \mathcal{N}_q$, if $r.choice \neq q$ in γ_t with $t \in [t_1..t_2]$, then $\forall t' \in [t..t_2]$, $r.choice \neq q$. So, there is a neighbor r of q that $\forall t \in [t_1..t_2]$, $r.choice = q$. Then, from the definition of $\texttt{ChosenCand}(r)$, $\forall t \in [t_1..t_2]$, the guard of `Vote` is *true* at r in γ_t. So, r executes (at least) one of the two first actions in the second round to set $r.choice$ to $\perp$, a contradiction.

(7) $\forall q \in \mathcal{N}_v, q.inA$ in γ_{t_3}.

Proof of Claim 7: Let q be a neighbor of v. By (2), $\forall t \in [t_2..t_3]$, $\texttt{CanLeave}(q) = false$. So, $\forall t \in [t_2..t_3]$, if $q.inA$ in γ_t, then $\forall t' \in [t..t_3]$, $q.inA$ in $\gamma_{t'}$. Hence, to show this claim, it is sufficient to show that $\exists t \in [t_2..t_3]$ such that $q.inA$ in γ_t. Assume the contrary: $\forall t \in [t_2..t_3]$, $\neg q.inA$. By (1) and (4), $\forall t \in [t_2..t_3]$, $\texttt{IsMissing}(q)$ holds in γ_t. Then, using (6), we deduce that the guard of the highest priority action of q, `Join`, is true in every configuration γ_t with $t \in [t_2..t_3]$. So, q joins the alliance in the third round, contradiction.

By (5), (7), and the fact that $\delta_v \geq g(v)$, $\texttt{Fga}(v)$ holds in γ_{t_3}, a contradiction. $\square$

By Remark 1, Lemmas 9, 12, and 13, we have the following:

Corollary 5 *If $f \geq g$, $\mathcal{MA}(f, g)$ is self-stabilizing* w.r.t. SP_{flc}*, and the first convergence time of $\mathcal{MA}(f, g)$ is at most four rounds.*

Lemma 14 *If $f \geq g$, then from any configuration where $\forall p \in V, \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$, Action `Join` is forever disabled at every process.*

Proof. Let γ by any configuration where $\forall p \in V, \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$. Then, $\texttt{Fga}(p)$ implies that $\neg p.inA \Rightarrow \texttt{NbA}(p) \geq f(p)$ in γ. Moreover, $(\forall q \in \mathcal{N}_p, \texttt{Fga}(q) \wedge \texttt{NbAOk}(q))$ implies $\neg\texttt{IsMissing}(p)$ in γ. So, Action `Join` is disabled at every process p in γ. By Lemma 10, we are done. $\square$

Lemma 15 *Let γ be any configuration where $\forall p \in V, \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$. If $f \geq g$, a configuration where $\forall p \in V, \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ is forever true is reached in at most one round from γ.*

Proof. By Lemmas 9 and 10, it is sufficient to show that $\forall p \in V$, there is a configuration in the first round starting from γ where $\texttt{NbAOk}(p)$ holds. Let p be a process. Consider the following two cases:

- *The value of p.inA changes during the first round from γ.* If p leaves A, then by Lemma 6, we are done. Otherwise, p executes `Join` in some step $\gamma' \mapsto \gamma''$ of the round. So, $\texttt{NbA}(p) \geq f(p)$ in γ' (Lemma 9) and consequently, $p.nbA \geq f(p)$ in γ''. As $f(p) \geq g(p)$ and $p.inA = true$ in γ'', we are done.

- *The value of* p.inA *does not change during the first round from* γ. Assume that $\texttt{NbAOk}(p) = false$ in all the configurations of the first round from γ. Then, as $\texttt{Fga}(p)$ is always *true* (Lemma 9), the guard of Action `Count` is always *true* during this round, and consequently p executes at least one of its three first actions in the round, in particular, $p.nbA \leftarrow \texttt{NbA}(p)$. Again, as $\texttt{Fga}(p)$ is always *true* during the round (Lemma 9), we obtain a contradiction, and thus we are done.

□

Lemma 16 *If $f \geq g$, then from any configuration where $(\forall p \in V, \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p))$, and A is not a 1-minimal (f, g)-alliance, at least one process permanently leaves A every five rounds.*

Proof. By contradiction. Let γ_{t_0} be a configuration where $\forall p \in V, \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$. Consider any execution (starting in γ_{t_0}) $e = \gamma_{t_0} \ldots \gamma_{t_1} \ldots \gamma_{t_2} \ldots \gamma_{t_3} \ldots \gamma_{t_4} \ldots \gamma_{t_5} \ldots$, where γ_{t_1}, γ_{t_2}, γ_{t_3}, γ_{t_4}, γ_{t_5} respectively are the last configurations of the first, second, third, fourth, fifth round of e. By Lemma 14, it is sufficient to show that $\exists t \in [t_0..t_5 - 1]$ such that some process leaves the alliance during $\gamma_t \mapsto \gamma_{t+1}$. Assume that no such a configuration exists.

Let $S = \{p \in V, p.inA \wedge \texttt{NbA}(p) \geq f(p) \wedge (\forall q \in \mathcal{N}_p, \texttt{HasExtra}(q))\}$. As A is not a 1-minimal (f, g)-alliance during the five first rounds after γ_{t_0}, $S \neq \emptyset$. Moreover, as no process leaves (by hypothesis) or joins (by Lemma 14) the alliance during the five first rounds from γ_{t_0}, S is constant during these rounds. Let $p_{\min} = \min(S)$.

We derive a contradiction, using the following six claims:

(1) $\forall t \in [t_1..t_5], \forall p \in V, p.nbA = \texttt{NbA}(p)$ in γ_t.

Proof of Claim 1: First, by hypothesis, $\forall p \in V$, the value of $\texttt{NbA}(p)$ is constant during the five first rounds. So, to show the claim, it is sufficient to prove that $\forall p \in V$, $\exists t \in [t_0..t_1]$, $p.nbA = \texttt{NbA}(p)$ in γ_t. Assume the contrary: there is a process p such that $\forall t \in [t_0..t_1]$, $p.nbA \neq \texttt{NbA}(p)$ in γ_t. Then, $\forall t \in [t_0..t_1]$, the guard of `Count` is *true* at p. As Action `Join` is disabled forever at p (by Lemma 14), p executes the second or third actions, in particular $p.nbA \leftarrow \texttt{NbA}(p)$, during the first round, and we obtain a contradiction.

(2) $\forall t \in [t_1..t_5]$, $\texttt{IsBusy}(p_{\min}) = false$ in γ_t.

Proof of Claim 2: From (1) and the definition of $p_{\min}$.

(3) $\forall t \in [t_2..t_5]$, $p_{\min}.choice = \bot$ in γ_t.

Proof of Claim 3: By (2) and the definition of $p_{\min}$, $\forall t \in [t_1..t_5]$, $\texttt{IamCand}(p_{\min})$ is *true* but $\texttt{MinCand}(p_{\min}) < p_{\min}$ is *false* in γ_t. So, $\forall t \in [t_1..t_5]$, $\texttt{ChosenCand}(p_{\min}) = \bot$ in γ_t. Hence to show the claim, it is sufficient to prove that $\exists t \in [t_1..t_2]$, $p_{\min}.choice = \bot$ in γ_t. Assume the contrary: $\forall t \in [t_1..t_2]$, $p_{\min}.choice \neq \bot$ in γ_t and consequently the guard of Action `Vote` is *true* in γ_t. Now, $\forall t \in [t_1..t_2]$, `Join` is disabled at $p_{\min}$ in γ_t by Lemma 14. So, $p_{\min}$ executes Action `Vote` during the second round, and we are done.

(4) $\forall t \in [t_2..t_5]$, $\neg p_{\min}.busy$ in γ_t.

Proof of Claim 4: By (2), if $\exists t \in [t_1..t_5]$ such that $\neg p_{\min}.busy$ in γ_t, then $\forall t' \in [t..t_5]$, $\neg p_{\min}.busy$ in $\gamma_{t'}$. Hence to show the claim, it is sufficient to prove that $\exists t \in [t_1..t_2]$ such that $\neg p_{\min}.busy$ in γ_t. Assume the contrary: $\forall t \in [t_1..t_2]$, $p_{\min}.busy = true$ in γ_t. $\forall t \in [t_1..t_2]$, `Join` and `Count` are disabled at $p_{\min}$ in γ_t (Lemma 14 and (1)). By (2), $\forall t \in [t_1..t_2]$, the guard of Action `Flag` is *true* at $p_{\min}$ in γ_t. Consequently, $p_{\min}$ executes `Vote` or `Flag` during the second round, and we are done.

(5) $\forall t \in [t_3..t_5], \forall q \in \mathcal{N}_{p_{\min}}, q.choice \in \{\bot, p_{\min}\}$ in γ_t.

Proof of Claim 5: By (4) and the definition of $p_{\min}$, $\forall t \in [t_2..t_5]$, $\forall q \in \mathcal{N}_{p_{\min}}$, $\texttt{ChosenCand}(q) = p_{\min}$ in γ_t. Hence, to show the claim, it is sufficient to prove that $\forall q \in \mathcal{N}_{p_{\min}}$, $\exists t \in [t_2..t_3]$ such that $q.choice \in \{\bot, p_{\min}\}$ in γ_t. Assume the contrary: let q be a neighbor of $p_{\min}$, and assume that $\forall t \in [t_2..t_3]$, $q.choice \notin \{\bot, p_{\min}\}$ in γ_t. Then, the guard of Action `Vote` is *true* at q in γ_t. Now, $\forall t \in [t_2..t_3]$, `Join` is disabled at q in γ_t, by Lemma 14. So, q executes Action `Vote` during the second round, and we are done.

(6) $\forall t \in [t_4..t_5], \forall q \in \mathcal{N}_{p_{\min}}, q.choice = p_{\min}$ in γ_t.

Proof of Claim 6: By (4) and the definition of $p_{\min}$, $\forall t \in [t_3..t_5]$, $\forall q \in \mathcal{N}_{p_{\min}}$, $\texttt{ChosenCand}(q) = p_{\min}$ in γ_t. Hence to show the claim, it is sufficient to prove that $\forall q \in \mathcal{N}_{p_{\min}}$, $\exists t \in [t_3..t_4]$, $q.choice = p_{\min}$ in γ_t. Assume the contrary: Let q be a neighbor of $p_{\min}$. Assume that $\forall t \in [t_3..t_4]$, $q.choice \neq p_{\min}$ in γ_t. Then, $\forall t \in [t_3..t_4]$, $q.choice = \bot$ in γ_t by (5) and consequently the guard of Action Vote is *true* at q in γ_t. Now, $\forall t \in [t_3..t_4]$, Join is disabled at q in γ_t, by Lemma 14. So, q executes Action Vote during the third round and we are done.

From γ_{t_0}, Action Join is disabled at $p_{\min}$ forever. By (3), (4), and the definition of $p_{\min}$, $\forall t \in [t_4..t_5]$ Action Vote is disabled at $p_{\min}$. By (1), $\forall t \in [t_4..t_5]$ Action Count is disabled at $p_{\min}$. By (2) and (4), $\forall t \in [t_4..t_5]$ Action Flag is disabled at $p_{\min}$. By (3), (6), and the definition of $p_{\min}$, $\forall t \in [t_4..t_5]$, Leave is enabled at $p_{\min}$. So, $p_{\min}$ leaves the alliance during the fifth round, contradiction. □

Theorem 2 *If $f \geq g$, $\mathcal{MA}(f,g)$ is silent and self-stabilizing* w.r.t. $SP_{1-Minimal}$ *and its stabilization time is at most $5n+8$ rounds.*

Proof. By Lemmas 12 through 16, starting from any configuration, the system reaches a configuration γ from which A is a 1-minimal (f,g)-alliance and Actions Join and Leave are disabled forever at every process, in $5n+5$ rounds. So, it remains to show that the system reaches a terminal configuration after at most three rounds from γ.

The following three claims establish the proof:

(1) After one round from γ, $\forall p \in V$, $p.nbA = \texttt{NbA}(p)$ forever.

Proof of Claim 1: From γ, for every process p, Join is disabled forever and $\texttt{NbA}(p)$ is constant. So, if necessary, p fixes the value of $p.nbA$ to $\texttt{NbA}(p)$ within the next round by Vote or Count.

(2) After two rounds from γ, $\forall p \in V$, $(p.inA \Rightarrow p.busy) \wedge p.busy = \texttt{IsBusy}(p)$ forever.

Proof of Claim 2: When the second round from γ begins, for every process p, values of $p.inA$ and $p.nbA$ are constant, moreover Join and Count are disabled forever at p (by hypothesis and claim (1)). So, if necessary, p fixes the value of $p.busy$ to $\texttt{IsBusy}(p)$ within the next round by Vote or Flag. Hence, after two rounds from γ, $\forall p \in V$, $p.busy = \texttt{IsBusy}(p)$ holds forever.

Finally, assume that there is a process p such that $p.inA \wedge \neg p.busy$ after two rounds from γ. Then, $p.inA \wedge \texttt{NbA}(p) \geq f(p) \wedge \texttt{IsExtra}(p)$. Now, by (1), this means that $p.inA \wedge \texttt{NbA}(p) \geq f(p) \wedge (\forall q \in \mathcal{N}_p, (\neg q.inA \Rightarrow \texttt{NbA}(q) > f(q)) \wedge (q.inA \Rightarrow \texttt{NbA}(q) > g(q)))$, which contradicts the fact that A is a 1-minimal (f,g)-alliance. Hence, after two rounds from γ, $\forall p \in V$, $(p.inA \Rightarrow p.busy)$ holds forever.

(3) After three rounds from γ, $\forall p \in V$, $p.choice = \bot$ forever.

Proof of Claim 3: When the third round from γ begins, for every process p, $\texttt{Cand}(p) = \emptyset$ forever by Claim (2), which implies that $\texttt{ChosenCand}(p) = \bot$ forever. Remember also that Join is disabled forever for every process. So, if necessary, p fixes the value of $p.choice$ to $\bot$ within the next round by Vote.

From the three previous claims, we can deduce that after at most three rounds from γ (that is, at most $5n+8$ rounds from the initial configuration), the system reaches a terminal configuration where $SP_{Minimal}$ holds, by Lemma 3. □

By Property 1, Corollary 5, and Theorem 2, we have:

Corollary 6 *If $f \geq g$, $\mathcal{MA}(f,g)$ is silent and safely converging self-stabilizing* w.r.t. $(SP_{flc}, SP_{Minimal})$, *its first convergence time is at most four rounds, its second convergence time is at most $5n+4$ rounds, and its stabilization time is at most $5n+8$ rounds.*

5 Conclusion and Perspectives

We have given a silent self-stabilizing algorithm, $\mathcal{MA}(f,g)$, that computes a minimal (f,g)-alliance in an asynchronous network with unique node IDs, assuming that $f \geq g$ and every process p has a degree at least $g(p)$. $\mathcal{MA}(f,g)$ is also *safely converging*: It first converges to a (not necessarily minimal) (f,g)-alliance in at most four rounds and then continues to converge to a minimal one in at most $5n+4$ additional rounds. We have verified correctness and time complexity of $\mathcal{MA}(f,g)$, assuming the weakest scheduling assumption: the distributed unfair daemon. Its memory requirement is $O(\log n)$ bits per process and its stabilization time in steps is $O(\Delta^3 n)$.

The immediate extension of our work is to try to reduce the stabilization time to $O(\mathcal{D})$ rounds. It would be interesting to study the (f,g)-alliance problem without the constraint that $f \geq g$. We conjecture that $\mathcal{MA}(f,g)$ is still self-stabilizing in that case. However, we already know that it does not guarantee a good safe convergence property in the case $f < g$: Indeed, in that case, any process can join A several times, giving us a round complexity of $\Omega(n)$ for convergence to a feasible legitimate configuration. We believe that when $f < g$, it is impossible to guarantee $O(1)$ round convergence to a feasible legitimate configuration, where a (not necessarily minimal) (f,g)-alliance is defined.

Our work is a step toward generalization of safe convergence to a wide class of problems.

References

[1] Edsger W. Dijkstra. Self-Stabilizing Systems in Spite of Distributed Control. *Commun. ACM*, 17:643–644, 1974. 1, 2.2

[2] Sukumar Ghosh, Arobinda Gupta, Ted Herman, and Sriram V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996*, pages 45–54. ACM, 1996. 1

[3] Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997. 1

[4] Shay Kutten and Boaz Patt-Shamir. Time-adaptive self stabilization. In James E. Burns and Hagit Attiya, editors, *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997*, pages 149–158. ACM, 1997. 1

[5] Hirotsugu Kakugawa and Toshimitsu Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. In *IPDPS*, 2006. 1, 1.2

[6] Christophe Genolini and Sébastien Tixeuil. A lower bound on dynamic k-stabilization in asynchronous systems. In *21st Symposium on Reliable Distributed Systems (SRDS 2002), 13-16 October 2002, Osaka, Japan*, pages 212–. IEEE Computer Society, 2002. 1

[7] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing approximation algorithm for the minimum weakly connected dominating set with safe convergence. In *Proceedings of the First International Workshop on Reliability, Availability, and Security (WRAS)*, pages 57–67, Paris, France, September 2007. 1

[8] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing 6-approximation for the minimum connected dominating set with safe convergence in unit disk graphs. *Theoretical Computer Science*, 428:80–90, 2012. 1

[9] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007. 1

[10] Anupam Gupta, Bruce M. Maggs, Florian Oprea, and Michael K. Reiter. Quorum placement in networks to minimize access delays. In Marcos Kawazoe Aguilera and James Aspnes, editors, *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC 2005, Las Vegas, NV, USA, July 17-20, 2005*, pages 87–96. ACM, 2005. 1

[11] Mitre Costa Dourado, Lucia Draque Penso, Dieter Rautenbach, and Jayme Luiz Szwarcfiter. The south zone: Distributed algorithms for alliances. In *SSS*, pages 178–192, 2011. 1.2, 2.4, 1

[12] Pradip K. Srimani and Zhenyu Xu. Distributed protocols for defensive and offensive alliances in network graphs using self-stabilization. In *ICCTA*, pages 27–31, 2007. 1.2

[13] Volker Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Inf. Process. Lett.*, 103(3):88–93, 2007. 1.2

[14] Guangyuan Wang, Hua Wang, Xiaohui Tao, and Ji Zhang. A self-stabilizing algorithm for finding a minimal k-dominating set in general networks. In Yang Xiang, Mukaddim Pathan, Xiaohui Tao, and Hua Wang, editors, *Data and Knowledge Engineering*, Lecture Notes in Computer Science, pages 74–85. Springer Berlin Heidelberg, 2012. 1.2

[15] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory Requirements for Silent Stabilization. In *PODC*, pages 27–34, 1996. 2.3